REMARKS

Introduction

Claims 1, 20, and 39-50 have been amended. The application continues to include claims 1-13, 20-32 and 39-50. Reconsideration of the rejection of the application is respectfully requested in view of the claim amendments and the following remarks.

Specification Objections

The specification is objected to because claims 39-50 recite "computer-readable medium." The Office Action states that Applicants' specification does not describe/support computer-readable medium. Applicants respectfully traverse this objection. The publication of the present application (i.e., U.S. Patent Publication No. 2004/0040011, or "Specification") describes a computer system, at paragraph [0071], where the computer system includes, in addition to other components, one or more processors 302, system memory 304, and mass storage devices 306, such as diskette, hard drive, CDROM, etc. *See* Specification at paragraph [0071]; Fig. 3. One of ordinary skill in the relevant art would appreciate that system memory 304 and mass storage devices 306 are each an example of a computer-readable medium, as they are each a type of a medium that is readable by the computer system. Accordingly, the term "computer-readable medium" is adequately described and supported in Applicants' specification. Thus, Applicants respectfully request that the objection be withdrawn.

The Claims are Directed to Statutory Subject Matter

Claims 39-50 are rejected under 35 U.S.C. §101 because the claimed invention is directed to non-statutory subject matter such as software per se. More specifically, the Office Action alleged that the term "computer-readable medium," given its broadest reasonable interpretation encompasses any medium readable by a computer, which includes carrier waves or signals. In response, claims 39-50 have been amended to replace the term "computer-readable medium" with "non-transitory computer-readable medium." Accordingly, the amendment to the claims effectively moots the rejection. Thus, Applicants respectfully request that the rejection be withdrawn.

The Claims are Allowable because the Prior Art Fails to Disclose Processing a Code Statement of the Second Programming Language within a First Code Section, and Processing a Code Statement of the First Programming Language within a Second Code Section, Where Both Code Statements are within a Data Processing Representation

Claims 1, 2, 3, 6, 7, 20, 21, 22, 25, 26, 39, 40, 41, 44 and 45 are rejected under 35 U.S.C. §102(e) as being anticipated by Wang, U.S. Patent No. 6,292,936 ("Wang"). Claims 4, 5, 8, 23, 24, 27, 42, 43 and 46 are rejected under 35 U.S.C. §103(a) as being unpatentable over Wang in view of Claussen et al., U.S. Patent No. 6,732,330 ("Claussen"). Claims 9-13, 28-32, and 47-50 are rejected under 35 U.S.C. §103(a) as being unpatentable over Wang in view of Conner et al., U.S. Patent No. 5,428,792 ("Conner"). Reconsideration of these rejections is respectfully requested because the prior art fails to disclose invoking a second code statement processing unit of a second programming language to process a code statement of the second programming

language within a first code section, and invoking a first code statement processing unit of a first programming language to process a code statement of the first programming language within a second code section, where both code statements are within a data processing representation.

In one embodiment, a computing environment is provided with an execution engine, supplemented with a number of language specific processing units to facilitate execution of data processing representations expressed with programming instructions of multiple programming languages. *See e.g.*, U.S. Patent Publication No. 2004/0040011 (*i.e.*, "Specification") at paragraph [0022]; Fig. 1. According to the embodiment, upon encountering a code section/statement of a language within a data processing representation, the execution engine invokes the corresponding language specific processing unit to augment and provide the language specific processing required to process and facilitate execution of the code section/statement. *See e.g.*, Specification at paragraph [0025]; Fig. 1. Upon encountering a sub-section written in an unknown programming language, within the data processing representation, the language specific processing unit can delegate the processing of the sub-section back to the execution engine. *See e.g.*, Specification at paragraph [0026]; Fig. 1. The execution engine, in turn, can pass the sub-section to an appropriate language specific processing unit and return the result to the requesting language specific processing unit. *See e.g.*, Specification at *id*. A data processing representation can include one or more code sections, where each section may include sub-sections written in one or more other languages, and each sub-section may, in turn, have sub-sub-sections written in

other languages. *See e.g.*, Specification at paragraph [0026]; Fig. 1. The execution engine, and corresponding language specific processing units, process the data processing representation as described above. *See e.g.*, Specification at paragraphs [0060]-[0063]; Fig. 2.

Wang describes a distributed computer system that includes server system 106 executing Web daemons 108. *See* Wang at col. 2, lines 36-48; FIG. 1. Each web daemon 108 includes runtime processors 110 and 112 (i.e., Java Virtual Machine (JVM) 110 that executes Java programming statements and a VisualBasic Script interpreter (VBSI) 112 that executes VisualBasic Script programming statements). *See* Wang at col. 2, lines 49-55; FIG. 1. The server system also includes one or more translators 114, where each translator 114 (e.g., HTML Parser 114) interprets an original input source 116 comprising scripts, programming instructions, etc., and translates the original input source 116 into two intermediate sources (i.e., intermediate source 200 and intermediate source 202). *See* Wang at col. 2, lines 56-67; col. 3, lines 25-30; FIGS. 1 and 2. To create the intermediate source 200, the HTML Parser 114 translates a first VisualBasic Script block in the original input source 116 into a thread object run method, a notify method, and an immediate wait method. *See* Wang at col. 3, lines 31-40; FIG. 2. The remaining VisualBasic Script blocks in the original input source 116 are translated into notify method and wait method invocations. *See* Wang at col. 3, lines 40-43; FIG. 2. To create the intermediate source 202, the HTML Parser 114 translates every HTML block after the first VisualBasic Script block into a synchronizer token. *See*

Wang at col. 3, lines 44-46. During runtime, the JVM 110 and the VBSI 112 execute the respective intermediate sources 200 and 202. *See* Wang at col. 3, lines 50-52.

Wang further describes that the JVM 110 performs its normal processing of the intermediate source 200 until it invokes a thread object run method to initiate execution of the VBSI 112, sends a notification to the VBSI 112, and then waits for a notification to be received from the VBSI 112. *See* Wang at col. 3, lines 57-67; FIG. 2. Once invoked, the VBSI 112 performs its normal processing of the intermediate source 202 until it encounters a synchronizer token, and, at that point, the VBSI 112 sends a notification to the JVM 110, and then waits to receive a notification from the JVM 110. *See* Wang at col. 4, lines 1-8; FIG. 2. Once the JVM 110 receives the notification from the VBSI 112, the JVM 110 continues its normal processing of the intermediate source 200; similarly, once the VBSI 112 receives the notification from the JVM 110, the VBSI 112 continues its normal processing of the intermediate source 202. *See* Wang at col. 4, lines 9-23; FIG. 2. The sequence of interaction between the JVM 110 and the VBSI 112 can be repeated indefinitely. *See* Wang at col. 4, lines 24-26; FIG. 2.

The Office Action took the position that the processing of the intermediate source 200 by the JVM 110 described in Wang discloses the element "invoking, by the execution engine, a second code statement processing unit of a second programming language to process a code statement, when the first code statement processing unit locates a code statement of the second programming language within the first code section, and invokes the execution engine recursively," of independent claim 1, and similar elements of independent claims 20 and 39. Likewise, the Office Action took the

position that the processing of the intermediate source 202 by the VBSI 112 discloses the element "invoking, by the execution engine, the first code statement processing unit of the first programming language to process a code statement, when the second code statement processing unit locates a code statement of the first programming language within the second code section, and invokes the execution engine recursively," of independent claim 1, and similar elements of independent claims 20 and 39. However, Wang explicitly discloses that a translator translates an original input source (that contains programming instructions) into two intermediate sources, where the first intermediate source includes programming instructions of a first language and place holders represents programming instructions of a second language, and where the second intermediate source includes programming instructions of the second language and place holders represents programming instructions of the first language. Thus, neither the JVM, nor the VBSI, process code sections that are both within the original input source. Instead, in Wang, a separate translation step is required that translates the original input source into two intermediate sources, where the JVM processes the first intermediate source, and the VBSI processes the second intermediate source. Therefore, Wang fails to disclose or suggest invoking a second code statement processing unit of a second programming language to process a code statement of the second programming language within a first code section, and invoking a first code statement processing unit of a first programming language to process a code statement of the first programming language within a second code section, where both code statements are within a data processing representation.

Furthermore, neither Claussen nor Conner cure the deficiencies of Wang. Claussen describes a page handling framework where different scripting languages may reside side-by-side, or nested within each other on the same web page. *See* Claussen at col. 2, lines 58-60. Claussen fails to disclose or suggest invoking a second code statement processing unit of a second programming language to process a code statement of the second programming language within a first code section, and invoking a first code statement processing unit of a first programming language to process a code statement of the first programming language within a second code section, where both code statements are within a data processing representation. Conner describes a system for defining language dependent object definitions as a neutral set of information from which object support for any language, including support between languages, is provided. *See* Conner at Abstract. Similar to Claussen, Conner fails to disclose or suggest invoking a second code statement processing unit of a second programming language to process a code statement of the second programming language within a first code section, and invoking a first code statement processing unit of a first programming language to process a code statement of the first programming language within a second code section, where both code statements are within a data processing representation.

In contrast to the cited prior art, amended independent claim 1 recites "invoking, by the execution engine, a second code statement processing unit of a second programming language to process a code statement, when the first code statement processing unit locates a code statement of the second programming language within

the first code section, and invokes the execution engine recursively," “invoking, by the execution engine, the first code statement processing unit of the first programming language to process a code statement, when the second code statement processing unit locates a code statement of the first programming language within the second code section, and invokes the execution engine recursively,” and “wherein the code statement of the second programming language within the first code section, and the code statement of the first programming language within the second code section, are both within the data processing representation.” For at least these reasons, amended independent claim 1, and amended independent claims 20 and 39, which recites similar limitations, should now be allowable over the cited prior art. The remaining claims depend from one of the above independent claims and should also be allowable for at least the above reasons.

Conclusion

Applicants respectfully request favorable action in connection with this application.

The Examiner is invited and urged to contact the undersigned to discuss any matter concerning this application.

No fee should be required for this submission. However, should any fee be required, the Commissioner is authorized to charge any such fee to Counsel's Deposit Account 50-2222.

Respectfully submitted,

Date: December 29, 2011

/Keith M. Mullervy/
Keith M. Mullervy
Attorney for Applicants
Registration No. 62,382

Customer No. 74739
SQUIRE, SANDERS & DEMPSEY (US) LLP
14TH Floor
8000 Towers Crescent Drive
Vienna, Virginia 22182-6212
Telephone: 703-720-7843
Email: keith.mullervy@ssd.com

KMM:sjm:sew